COMPSCI 687 Fall 2024 - Final Report

Kevin Dao

Prioritized Sweeping

1 Overview

Prioritized Sweeping, as it name suggests, allows efficient planning in reinforcement learning by focusing on state-action pairs that are expected to yield the greatest "impact". Impact here refers to the expected magnitude of change in the value function in response to new information. Because of the propagated effects on the value estimates of predecessor states, if the value of a state changes significantly, it may cause changes in the value estimates of other states that transition into it. Unlike uniform updates, Prioritized Sweeping prioritizes updates based on "urgency" through the use of priority queue, enabling fast propagation of value changes and allowing efficient use of computational resources.

2 Pseudocode

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow policy(S,Q)$
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Model(S,A) \leftarrow R, S'$
- (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
- (f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
- (g) Loop repeat n times, while $PQueue$ is not empty:
  - $S,A \leftarrow first(PQueue)$
  - $R,S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  - Loop for all $\bar{S},\bar{A}$ predicted to lead to S:
    - $\bar{R} \leftarrow$ predicted reward for $\bar{S},\bar{A},S$
    - $P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
    - if $P > \theta$ then insert $\bar{S},\bar{A}$ into $PQueue$ with priority P

Prioritized Sweeping is different from algorithms that has been covered in class because of its use of a "Model" function, which stores the reward and next state explored given a state and an action, as well as a Priority Queue, a well known property of search algorithms such as Dijkstra's.

To get the action A using the policy on state S and Q-value function Q, I opted for a soft-max approach that allows for continuous small exploration.

Prioritized Sweeping computes the priority using a modified TD-learning approach where instead of using state value function for its computation, it uses state-action value function with the value

maximizing action. If the priority exceed a threshold theta, then it is inserted into the priority queue.

The Q-value function update considers the first "n" elements of the priority queue, updating the corresponding Q-value function for state-action pair using the already computed priority and a step size alpha. Changes are propagated to parent state-action pairs, where its priority is computed and considered to add into the priority queue.

3 Hyperparameters Fine-tuning

There are 6 hyperparameters that I am fine-tuning for, those are:

- theta (θ) : priority threshold
- alpha (α) : step-size
- epsilon (ϵ) : exploration rate
- n : number of q value update
- niter : number of episodes
- episode length : number of steps per episode

Leveraging the hyperparameter search that we have done in Homework 2, I used Evolution Strategy to find the best set of parameters for each particular domain, Cat vs Monsters and 687-Gridworld.

There are 3 main parts in any Evolution Strategy which are population generation, fitness evaluation, and mutation.

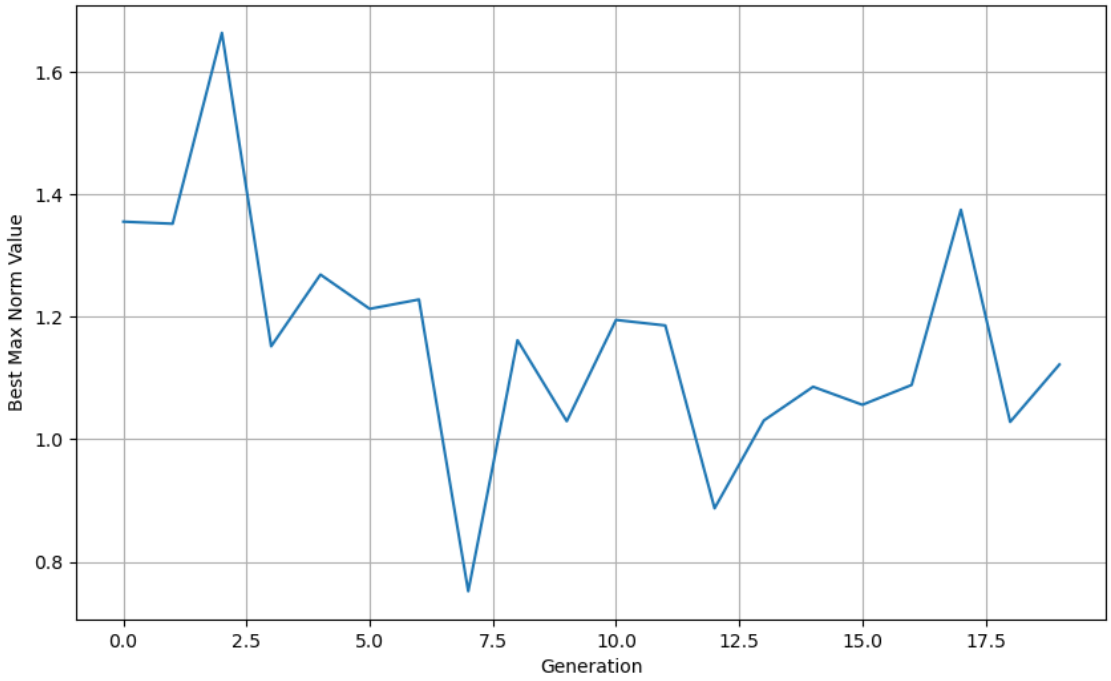
For population generation, I drew a uniform distribution of continuous variables such as theta, alpha and epsilon and drew random integers for discrete variables such as n, niter, and episode length. Because I want my evolution strategy to finish in reasonable time and exploiting the fact that most hyperparameters have bound, I set bounds for each hyperparameter, where theta, alpha, and epsilon ranges from 0.01 to 1, n ranges from 1 to 100, and niter and episode length ranges from 50 to 400.

For fitness evaluation, I uses the max norm value of the state value function of interest (converted from the outputted state-action value function) and the optimal value function (gotten from running Value Iteration).

For mutation, I sampled a normal distribution where the mean is 0 and the standard deviation is the difference in the bound for the respective hyperparameter times a constant "mutation strength". For discrete variables, I changed my mutation slightly, where the noise is rounded and divided by 10 to prevent explosion of hyperparameters, especially for large bounds.

4 Learning Curves

Cat vs Monsters' Learning Curve

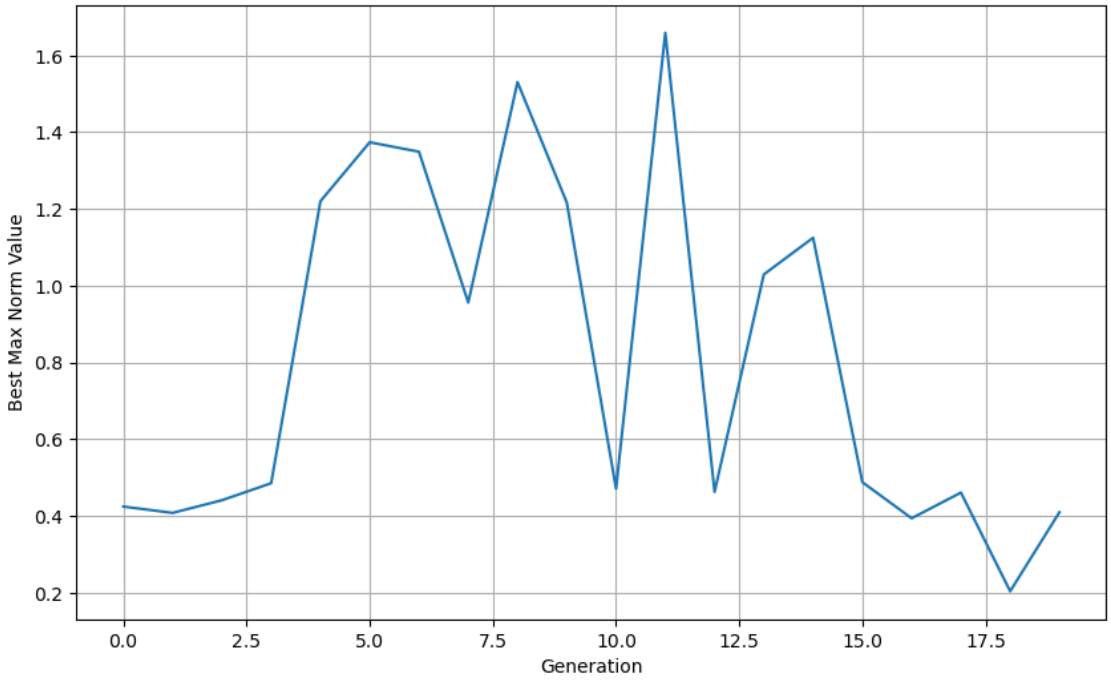


The final parameters found were:

- $\theta = 0.8384$
- $\alpha = 0.0735$
- $\epsilon = 1.0$
- $n = 8$
- niter $= 329$
- episode length $= 370$

The best Max-Norm Value was 0.7518

687-Gridworld's Learning Curve



The final parameters found were:

- $\theta = 0.8367$
- $\alpha = 0.8425$
- $\epsilon = 1.0$
- $n = 52$
- niter $= 249$
- episode length $= 168$

The best Max-Norm Value was 0.2038